МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по практической работе №4

по дисциплине «Алгоритмы и структуры данных»

Тема: кодирование и декодирование, БДП, хеш-таблицы, сортировки

Студент гр. 8304 _____ Ястребов И.М.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2019

Цель работы

Изучить основные способы кодирования и декодирования, сортировки, представления данных в виде БДП и хеш-таблиц

Задание

По заданному файлу F (типа file of Elem), все элементы которого различны, построить Рандомизированное БДП.

Для построенной структуры данных проверить, входит ли в неё элемент e типа Elem, и если входит, то удалить элемент e из структуры данных. Предусмотреть возможность повторного выполнения с другим элементом.

Описание алгоритма

1. Из консоли/файла считывается строка.
2. Исходя из предположения о корректности, на основе полученных данных в памяти формируется Treap-структура
3. Помним последний добавленный y(k), добавляем y(k+1)
4. y(k+1) > y(k) => y(k+1) = right(y(k))
5. Иначе идем вверх пока y(l) <y(k+1)
6. y(k+1) = right(y(l))
7. y(l).pre_right = y(k+1).left
8. Пользователь называет ключи элементов, они удаляются и текущее состояние дерева демонстрируется пользователю

Описание основных функций

```
template<typename elem, typename priority>
void printtree(nodePtr<elem, priority>& head) – отрисовка дерева в порядке КЛП;

void createIntNodeVector(std::vector<nodePtr<int, int>>& pairs, std::string& input) – формирование вектора узлов, которые далее располагаются в памяти

template<typename elem, typename priority>
class Node – класс узла Декартова дерева

static void split(nodePtr<elem, priority>, elem, nodePtr<elem, priority> &, nodePtr<elem, priority> &);

static void insert(nodePtr<elem, priority> &, nodePtr<elem, priority>);
```

```
	static void merge(nodePtr<elem, priority> &, nodePtr<elem, priority>, nodePtr<elem, priority>);

	static bool erase(nodePtr<elem, priority> &, elem); - функции работы с Декартовым деревом – вставка, вспомогательные split и merge, удаление erase
```

Вывод.

Был получен опыт работы с Treap-структурами данных.

ПРИЛОЖЕНИЕ А

Тестирование программы

```
1 2 3 4 5 6 7 8 9 10 11 12
tree :
((3;31650)((2;30968)((1;12152)##)#)((8;27913)((4;18903)#((6;17573)((5;16780)##)((7;2337)##)))((10;26961)((9;7471)##)((12
;25430)((11;24221)##)#))))
Which elements to delete?
3
found and removed 3
tree :
((2;30968)((1;12152)##)((8;27913)((4;18903)#((6;17573)((5;16780)##)((7;2337)##)))((10;26961)((9;7471)##)((12;25430)((11;
24221)##)#))))What elements to delete?
5
found and removed 5
tree :
((2;30968)((1;12152)##)((8;27913)((4;18903)#((6;17573)#((7;2337)##)))((10;26961)((9;7471)##)((12;25430)((11;24221)##)#))
))What elements to delete?
9
found and removed 9
tree :
((2;30968)((1;12152)##)((8;27913)((4;18903)#((6;17573)#((7;2337)##)))((10;26961)#((12;25430)((11;24221)##)#))))What elem
ents to delete?
12
found and removed 12
tree :
((2;30968)((1;12152)##)((8;27913)((4;18903)#((6;17573)#((7;2337)##)))((10;26961)#((11;24221)##))))What elements to delet
e?
6
found and removed 6
tree :
((2;30968)((1;12152)##)((8;27913)((4;18903)#((7;2337)##))((10;26961)#((11;24221)##))))What elements to delete?
```

ПРИЛОЖЕНИЕ Б

Файл Treap.hpp

```
#pragma once
#include <memory>

template<typename elem, typename priority>
class Node;

template<typename elem, typename priority>
using nodePtr = std::shared_ptr<Node<elem, priority>>;

template<typename elem, typename priority>
class Node {
public:
	elem key;
	priority prior;

	nodePtr<elem, priority> left, right;

	Node() = default;

	//default copy constructor and operator= are intended

	~Node() = default;

	Node(elem key, priority prior) : key(key), prior(prior), left(nullptr),
right(nullptr) { }

	static void split(nodePtr<elem, priority>, elem, nodePtr<elem, priority> &,
nodePtr<elem, priority> &);

	static void insert(nodePtr<elem, priority> &, nodePtr<elem, priority>);

	static void merge(nodePtr<elem, priority> &, nodePtr<elem, priority>, nodePtr<elem,
priority>);

	static bool erase(nodePtr<elem, priority> &, elem);
};

template<typename elem, typename priority>
void Node<elem, priority>::split(nodePtr<elem, priority> head, elem key, nodePtr<elem,
priority> & left, nodePtr<elem, priority> & right) {
	if (!head) {
		left = nullptr;
		right = nullptr;
	}

	else if (key < head->key) {
		split(head->left, key, left, head->left);
		right = head;
	}

	else {
		split(head->right, key, head->right, right);
		left = head;
	}
}

template<typename elem, typename priority>
void Node<elem, priority>::insert(nodePtr<elem, priority> &t, nodePtr<elem, priority> it) {
	if (!t)
		t = it;
```

```
	else if (it->prior > t->prior) {
		split(t, it->key, it->left, it->right);
		t = it;
	}

	else
		insert(it->key < t->key ? t->left : t->right, it);
}

template<typename elem, typename priority>
void Node<elem, priority>::merge(nodePtr<elem, priority> & t, nodePtr<elem, priority> l,
nodePtr<elem, priority> r) {
	if (!l || !r)
		t = l ? l : r;

	else if (l->prior > r->prior) {
		merge(l->right, l->right, r);
		t = l;
	}

	else {
		merge(r->left, l, r->left);
		t = r;
	}
}

template<typename elem, typename priority>
bool Node<elem, priority>::erase(nodePtr<elem, priority> & t, elem key) {
	if (!t)
		return false;

	if (t->key == key) {
		merge(t, t->left, t->right);

		return true;
	}

	return erase(key < t->key ? t->left : t->right, key);

}
```

Содержание Lab5.cpp

```
#include "pch.h"
#include <iostream>
#include <fstream>
#include "Treap.hpp"
#include <vector>
#include <string>
#include <ctime>

template<typename elem, typename priority>
void printtree(nodePtr<elem, priority>& head) {
	if (!head) {
		std::cout << '#';
		return;
	}
	std::cout << '(';
	std::cout << "(" << head->key << ";" << head->prior << ")";
	printtree(head->left);
	printtree(head->right);
	std::cout << ')';
}
```

```
void createIntNodeVector(std::vector<nodePtr<int, int>>& pairs, std::string& input) {
	srand((unsigned int)time(0));

	size_t index = 0;

	while (input[index] == ' ')
		++index;

	while (index < input.length()) {

		auto element = std::make_shared<Node<int, int>>(std::stoi(in-
put.substr(index)), rand() % INT_MAX);

		pairs.push_back(element);

		while (isdigit(input[index]))
			++index;
		while (input[index] == ' ')
			++index;
	}
}



int main(int argc, char* argv[]) {
	std::string input;

	if (argc == 2) {
		//freopen(argv[1], "r", stdin);
	}
	std::getline(std::cin, input);

	//example for <int, int>

	std::vector<nodePtr<int, int>> pairs;
	createIntNodeVector(pairs, input);

	nodePtr<int, int> head = nullptr;

	for (size_t i = 0; i < pairs.size(); ++i)
	{
		Node<int, int>::insert(head, pairs[i]);
	}

	std::cout << "tree : " << std::endl;

	printtree(head);

	std::cout << std::endl << "Which elements to delete?" << std::endl;

	int tmp = 0;

	while (std::cin >> tmp) {
		if (Node<int, int>::erase(head, tmp)) {
			std::cout  << "found and removed " << tmp << std::endl;

			std::cout << "tree : " << std::endl;

			printtree(head);
		}

		else {
			std::cout << "didin't find " << tmp << std::endl;

			std::cout << "tree : " << std::endl;
```

```
                printtree(head);
            }

            std::cout << "What elements to delete?" << std::endl;
        }

        return 0;
}
```